

POE User Guide

Table of Contents

Preface	3
Introduction	4
What POE Is	5
What POE Does	5
POE Terminology	6
Impact on Programming	9
Understanding SP Configuration	11
Local Reporting Tools	12
JS (Pool Summary)	12
JU (Node Usage)	12
JJ (Job Features)	13
LLSTATUS (Node Features)	14
SPJSTAT (Job Status)	15
Establishing Authorization	17
Compiling and Linking Parallel Programs	18
Relevant Compilers	18
Relevant Compiler Options	20
Setting Up the Execution Environment	22
Setting POE Environment Variables	23
Local Defaults	24
Basic POE Environment Variables	25
Tasks per Node	25
Node Allocation	27
Task Communications	28
Examples Using Basic POE Variables	29
Other POE Environment Variables	30
Enabling Large-Memory Pages on UM, UV, UP	38
Running Under POE	40
Execute Lines	40
Job Termination	42
Pitfalls	43
Parallel Performance Benchmarks	45
Disclaimer	47
Keyword Index	48
Alphabetical List of Keywords	50
Date and Revisions	51

Preface

- Scope: This guide introduces the role and background concepts for IBM's Parallel Operating Environment (POE), then explains POE's impact on programming, how to discover relevant features of your SP system configuration, how to compile parallel programs for use under POE, how to set up the execution environment appropriately, and how to run programs (parallel and serial) under POE (with warnings about known pitfalls). Some sections of this guide closely reflect corresponding sections of a tutorial on POE prepared and presented to LC users by Blaise Barney.
- Availability: POE is an IBM product available at LC on IBM SP machines (UP, UV, UM, and Purple).
- Consultant: For help contact the LC customer service and support hotline at 925-422-4531 (open e-mail: lc-hotline@llnl.gov, secure e-mail: lc-hotline@pop.llnl.gov).
- Printing: The print file for this document can be found at

OCF: <http://www.llnl.gov/LCdocs/poe/poe.pdf>

SCF: https://lc.llnl.gov/LCdocs/poe/poe_scf.pdf

Introduction

This POE Users Guide introduces the key features of IBM's Parallel Operating Environment (POE), the framework for developing and running parallel programs on IBM SP machines such as UP, UV, UM, and Purple.

This section explains what POE is and summarizes its role in managing parallel programs. Other sections of this guide then discuss programming constraints, how to discover relevant SP configuration features, compiling parallel programs, setting up the execution environment (by using relevant environment variables), and running your parallel code on the IBM machines.

IBM publishes much that directly or indirectly explains POE behavior. LC's IBM Documentation Directory is a good place to find comparative summaries of the most LC-relevant vendor publications, with links to online sources. Consult this open-network URL:

OCF: <http://www.llnl.gov/LCdocs/ibmdir>

As Linux (the open-source version of UNIX) becomes the most important operating system on LC's nonASCII production computers, AIX/POE users may have questions about AIX/Linux differences and compatibilities. While proprietary IBM (AIX) system software (such as POE) will clearly not be available under Linux, the status of relevant locally developed utilities (such as HTAR) and even of "standard" but differently implemented UNIX tools (such as CP) is often not easy to predict. POE users concerned about moving code to or from LC's Linux systems should therefore consult the Linux Differences (URL: <http://www.llnl.gov/LCdocs/linux>) guide for an overt, systematic review of important Linux features at LC. For a summary of the design goals and service enhancements added to LC's local version of Linux, consult the manual CHAOS: Linux from Livermore. (URL: <http://www.llnl.gov/LCdocs/chaos>) For information on SLURM, the resource manager that takes the place of LoadLeveler on LC's Linux (CHAOS) clusters, consult the SLURM Reference Manual. (URL: <http://www.llnl.gov/LCdocs/slurm>)

Other high-performance computer centers also run IBM SP machines with POE and also have developed local guidebooks of procedural and programming advice. One such book that is publicly shared is Juha Haataja and Tiina Kupila (Eds.), *Guide to the IBM SP Supercomputer* (2nd ed., 2001, 102 pages), published by CSC (the Finnish national scientific computing center), and available at

Overview: <http://www.csc.fi/oppaat/ibmsp>
PDF text: <http://www.csc.fi/oppaat/ibmsp/cscibmsp.pdf>

What POE Is

The Parallel Operating Environment (POE) is a distributed-memory, message-passing-based system designed to manage parallel programs on IBM RS/6000 machines that use the AIX operating system. At LC, POE is the user environment for the IBM SP computers (UP, UV, UM, and Purple). POE is part of IBM's Parallel Environment (PE) software product, and for most purposes the terms POE and PE are synonymous.

POE consists of a mix of closely related software tools, including:

- Ways to manage your parallel execution environment (special environment variables and corresponding command-line flags to set them).
- The Message Passing Interface (MPI) library for interprocess communications.
- Scripts for parallel compiling and linking.
- Parallel file-copy utilities and reporting tools, and optional profilers and run-time analysis tools (often with limited availability).

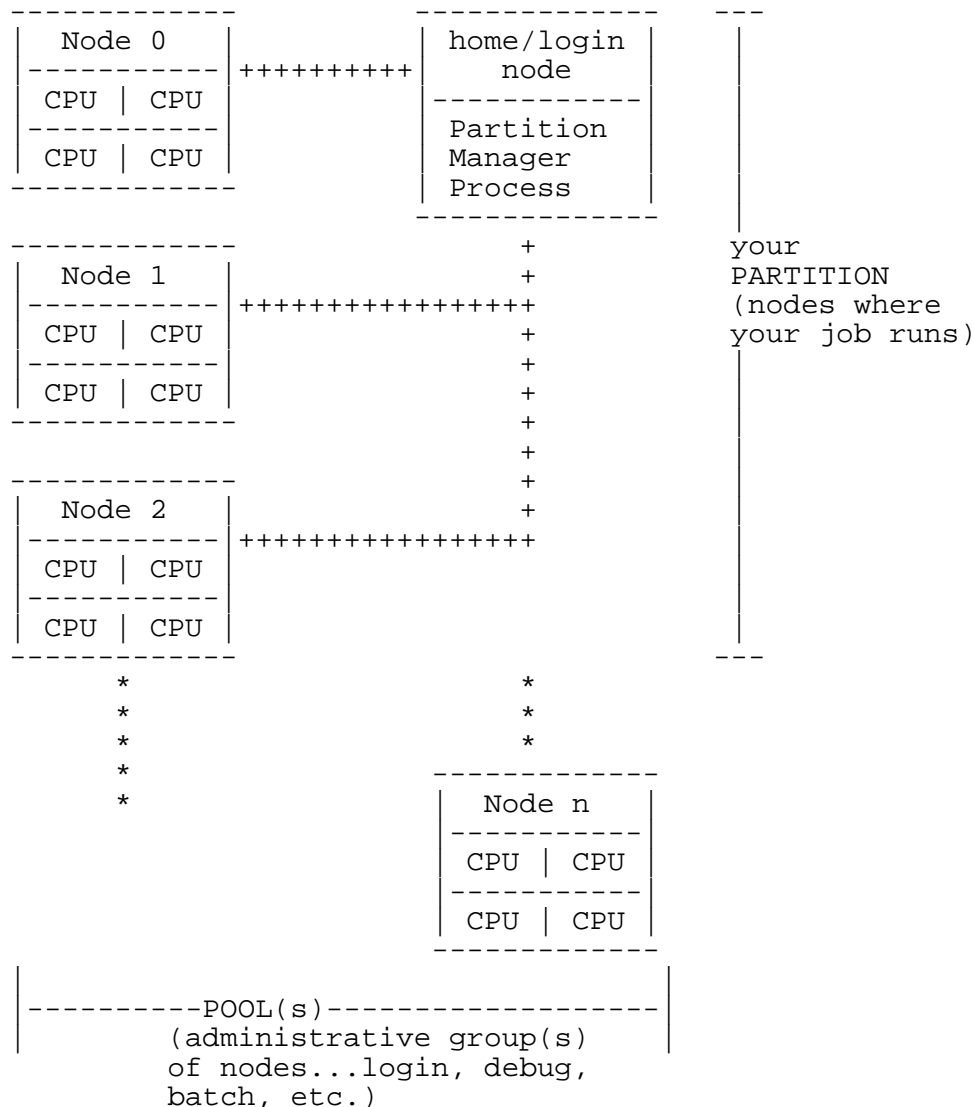
What POE Does

POE performs many tasks, some transparent to the user, that include:

- Linking the necessary parallel libraries during compilation (via parallel compiler scripts).
- Finding and acquiring the machine nodes for your parallel job (creating your job's "partition").
- Loading your executable onto all nodes acquired for your job (or, if the job is MPMD, loading multiple executables).
- Handling all input, error messages, and output (STDIN, STDERR, STDOUT) between the nodes of your parallel job.
- Handling all signals for the tasks in your job.
- Providing intertask communication and managing your job's use of processors and network adaptors.
- Retrieving system and job status information when requested, including error messages.
- (Optionally) running serial jobs and shell commands concurrently across many nodes on an SP machine.

POE Terminology

This diagram and the definitions below it explain and show the relationships between the most important POE terms and concepts:



- Node** is a single "machine," with a unique network name and address. In the IBM SP environment, each node is a "symmetric multiprocessor" (SMP) that has four CPUs, with shared memory and local disk space (but no memory is shared between nodes).
- Pool** is an administrative collection of nodes assigned by the SP system manager. Pools separate nodes into disjoint sets, each of which is used for a specific computational purpose (testing, batch runs, classes, etc.). On LC machines, the batch pool for production runs is by far the largest pool of nodes.

Partition	is the group of nodes on which your parallel program actually runs. Typically, at any time multiple partitions are active for the jobs of multiple users spread across an SP system. Theoretically, a node can be shared among several partitions (but this is most likely only if a gang scheduler is enforcing time sharing on the system).
Home	is the node where you log in and where you start your POE job (interactive use only). A remote node is any nonhome node in your partition.
Job Manager	<p>is a service traditionally provided by LoadLeveler, IBM's commercial batch scheduling software. LoadLeveler communicates with your job's partition manager (next item) to find and allocate needed nodes, to keep track of the switch, and to (optionally) enable jobs to use multiple CPUs on the same node.</p> <p>(On LC's Linux clusters, this role is performed by <u>SLURM</u> (URL: http://www.llnl.gov/LCdocs/slurm).) Starting in 2006, LC began gradually <i>replacing</i> LoadLeveler with SLURM even on production AIX machines, both open and secure. This often changes the node-reporting or job-reporting tools available to support POE jobs running under AIX. When SLURM utilities take the place of POE/AIX tools, those changes are noted in the appropriate tool sections below, and links to the relevant section of the SLURM Reference Manual are included.</p>

Partition Manager

is a daemon process that resides on your home node, starts automatically whenever you run an interactive POE job, and oversees its parallel execution. The partition manager generally operates transparently to the user, and it performs such tasks as:

- Obtains the nodes needed to run your parallel job.
- Establishes socket connections with each node in your partition.
- Sets up your user environment (as you specified in environment variables or command-line flags).
- Starts a process (called PMD) on each node in your partition. PMD interacts with the partition manager and actually becomes the parent process of your executable.
- Dynamically links your specified communications library to your executable.
- Loads your executable from the home node to each node in your partition.
- Routes STDIN, STDERR, and STDOUT on your home node to all other nodes in your partition, and exchanges signal and other control information with the PMD process on each node.

Protocol	specifies how the tasks of your job communicate. There are two choices:
User Space	is a fast method for intertask MPI communication, but can be used only with the high-performance switch (often called US protocol).
Internet	is a slower but more flexible method for intertask MPI communication (can be used with network adapters besides the high-performance switch (often called IP protocol)).

Node Allocation

is the process of (the job manager, LoadLeveler or SLURM) selecting the nodes on which your job will run. There are two choices:

Nonspecific	is the usual (and LC default) method of node allocation, in which you allow the system to do all node selection. Nonspecific node allocation is standard practice for batch jobs.
Specific	enables you to explicitly choose which nodes will run your POE job (by specifying a list of actual node names to use). Nonspecific node allocation is preferred practice on LC machines.

Impact on Programming

POE imposes some constraints on every application programmer, and ignoring these restrictions can cause your code to behave unpredictably or to fail unexplainably when it runs. The prime programming constraints from POE include:

- You should let POE handle signals whenever possible. All compiler scripts link in POE's own signal handlers, and these cover most signals that could cause program termination. POE uses its own signal handlers so it can terminate entire parallel jobs in an orderly manner if one task fails, and so that it can complete pending tasks (such as making trace files) successfully before terminating.
- POE requires special `exit()` and `atexit()` routines, so you should not code your own.
- You must pass to your program any string containing blanks by enclosing it in quotes and then "escaping" each quote with a preceding backslash. For example:

```
poe myprog \"this is one argument\"
```

- You should avoid large amounts of standard input or output because these can overload the partition manager process on your home node by exhausting its buffers.
- The POE signal handlers will cause some system routines to fail if your program uses the MPI signal-handling library. The following library and system calls will be interrupted and will not complete normally:

```
AIX msg routines
accept
aio_read/aio_write/aio_suspend
connect
exec/execv
fork
msem_lock/semop
open/close/fopen/fclose
pause
poll
recv/recvfrom/recvmmsg
select
send/sendto/sendvmsg
sleep/usleep/nsleep
system
```

- If you use the MPI threaded library (now the default), you are responsible for insuring that other linked libraries are thread safe and for making `pthread.h` the first include file.
- You must call `MPI_INIT` only once per task (not on each thread of that task), and you must call `MPI_FINALIZE` on the same thread that called `MPI_INIT`.
- To see the current system-imposed (configuration) limits (for file size, data size, stack size, and memory) on your programs on any LC IBM machine with POE, execute this reporting utility:

```
ulimit -a
```

- If you use mathematical library LIBM, which IBM provides under AIX, you may be able to optimize your code by instead using "tuned alternatives" to many LIBM functions from another library that IBM calls the "Mathematical Acceleration Subsystem" or MASS (libmass.a). For details see IBM's documentation at

<http://techsupport.services.ibm.com/server/mass?fetch=home.html>

Understanding SP Configuration

After preparing your parallel job but before compiling or running it, you should explore configuration and job behavior on the SP system where you plan to work. The following sections suggest practical ways to do this on LLNL's ASC SP machines, which may differ significantly from SP systems used for other purposes at other locations.

On every IBM SP system at LLNL is a plain text file called

```
/usr/local/docs/job.limits
```

that you can view with MORE, MOLE, or any text editor. The format differs from one machine to another, but this file always contains up-to-date technical details on system configuration for the specific machine where it resides. Included are

- the available job pools and their differences,
- the maximum number of nodes and CPUs (which may vary by time of day or by job pool),
- the maximum number of active jobs in any state (per user),
- the maximum number of tasks currently allowed per job and per node, and
- the current theoretical and practical limits on memory per node on this machine.

The same job-limit data in a tabular summary that allows easy comparison among different machines is available (with OTP authentication) at the OCF web site:

<https://lc.llnl.gov/computing/status/limits.html>

Besides such local (machine-specific) job limits, your job may encounter three broader, across-machine "resource partition limits" enforced by the LC batch system (LCRM). See the "Resource Partition Limits" section of LC's LCRM (DPCS) Reference Manual (URL: <http://www.llnl.gov/LCdocs/dpcs>) for an analysis of how these limits can affect your job's scheduling in unpredictable ways (and for how PSTAT reports each one). See the BRLIM section in the Bank and Allocation Manual (URL: <http://www.llnl.gov/LCdocs/banks>) for tips on using the utility that reports these global, partition-wide job limits.

For dynamic reports on machine features and jobs, see the next subsection on "Local Reporting Tools."

Local Reporting Tools

Several utility programs (some SP standards and some local to LLNL's ASC systems) provide up-to-date online information on your system's computing environment, status, and configuration. Five of the most useful tools are described in the following subsections, in the order in which you might typically run them (JS, JU, JJ, LLSTATUS, and SPIJSTAT).

JS (Pool Summary)

JS was developed at LLNL for the ASC systems to summarize the current SP node pools. JS takes no arguments, ends automatically, and reports

- The name of each pool (e.g., pdebug).
- The size of each pool (e.g., 8 nodes).
- The names of the nodes in each pool (e.g., white337-344).

On AIX machines where SLURM has replaced LoadLeveler as the underlying batch system, JS has been replaced by SINFO (URL: <http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.4.5>), run without options. See the SLURM Reference Manual for SINFO details and examples.

JU (Node Usage)

JU was developed at LLNL for the ASC systems to summarize the current SP node usage and availability. JU takes no arguments, ends automatically, and reports for each node pool on the SP system (sequentially by pool name):

- The name of the pool.
- The total nodes in the pool.
- The number of nodes now declared down.
- The number of nodes now in use.
- The number of nodes still available.
- The names of jobs now running in that pool and the node count for each job (e.g., rrygi-48).

JU works with both LoadLeveler and SLURM batch systems on LC's AIX machines.

JJ (Job Features)

JJ takes no arguments, ends automatically, and reveals the existence and reports some of the characteristics of every currently running job on the SP system, including both POE (interactive) and batch (LoadLeveler) jobs. Job reports are sorted numerically by jobid, but note that JJ knows nothing about the identification numbers assigned by PSUB and reported by PSTAT for LCRM jobs, so it uses an entirely different (nonLCRM) identifier for every job that it reports. For each current job, JJ lists

- Its own (nonLCRM) jobid.
- The job's owner (called "user").
- The date and time when the job started running.
- Whether the job is POE (interactive) or LoadLeveler (batch) managed.
- Whether the CPUs and network adapters used by the job are DEDICATED exclusively to it or available to be SHARED by other jobs.
- How many nodes the job uses, and the names of those nodes (e.g., white337-340).

On AIX machines where SLURM has replaced LoadLeveler as the underlying batch system, JJ has been replaced by SQUEUE (URL: <http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.3>), whose options control which job features are reported. See the SLURM Reference Manual for SQUEUE details and examples.

LLSTATUS (Node Features)

LLSTATUS is a LoadLeveler command that reports the current features of every SP node, or optionally reports just on specific nodes (if you supply their node names) or on specific features if you name the features using the -f (fixed-length fields) or -r (variable-length fields) control options (see the LLSTATUS man page for esoteric details). You execute LLSTATUS by typing

```
/usr/lpp/LoadL/full/bin/llstatus [-l] [nodenames]
```

where

- | | |
|------------------|--|
| -l | reports a long list of node features (several dozen). The default is a short list of 9 features, including down/available status, active/idle status, current load average, and operating system. Users of -l should probably redirect the LLSTATUS output to a file because it will be very long. |
| <i>nodenames</i> | is a blank-delimited list of target nodes on which to report. You can use the short name (white337) rather than the long name (white337.pacific.llnl.gov) of any node. The default (with no list of nodes) is a report on all nodes on the SP system, usually a very long list. |

On AIX machines where SLURM has replaced LoadLeveler as the underlying batch system, LLSTATUS has been replaced by SINFO (URL: <http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.4>), whose options control which node features are reported. See the SLURM Reference Manual for SINFO details and examples.

SPJSTAT (Job Status)

SPJSTAT was developed at LLNL for the ASC (IBM SP) systems. SPJSTAT is intended to supplement PSTAT, LC's primary job-monitoring tool, by revealing more about job/node interactions than appears in PSTAT reports. There are three different ways to use SPJSTAT.

BASIC REPORT.

If run without arguments, SPJSTAT ends automatically and reports for every (batch) job under LCRM control on the current cluster:

- The job's "native" (*not* LCRM/PSUB) identifier. On IBM SP systems, the native identifier is the "LoadLeveler (LL) Batch ID" (of the form white200.8392.0 rather than something like batch_6 or jetjob24, as PSUB assigns and PSTAT reports).
- The (login) name of the job's owner.
- The number of nodes used (e.g., 16) and the name of the job's master node (e.g., white341).
- The job's node pool (e.g., pbatch) and LCRM job class (e.g., normal).
- The job's current execution status (reported using one of these one- or two-letter codes, *not* LCRM job-status descriptors as with PSTAT):

```
C   is in the process of being removed.
H   has been deferred (held).
I   is idle.
NQ  has not been queued to run.
O   has both a system and user hold.
PM  has been preempted.
R   is running now.
S   has been held by the system.
ST  is starting.
U   has been held by the user.
W   is waiting to run.
```

WHAT-IF REPORT.

An alternative way to run SPJSTAT has the form:

```
spjstat -ln nodes -c constraints -cpn cpn
```

where

- | | |
|--------------------|--|
| <i>nodes</i> | specifies the number of nodes to be requested by a planned job (e.g., 32). |
| <i>constraints</i> | specifies any of the PSUB -c constraints to be requested for a planned job (e.g., Mb). |
| <i>cpn</i> | specifies the number of CPUs per node to be requested for a planned job (e.g., 4). |

SPJSTAT interprets this execute line with arguments as a request to check if a planned job requiring the specified nodes and constraints would actually run on the local system. It responds with a one-line message of the form "This job would [not] run" (along with a current status report as background).

POOL-ONLY REPORT.

Finally, if you run SPJSTAT with only the -p control option

```
spjstat -p
```

you get a report only on the LCRM "scheduling pool" for the cluster on which you execute SPJSTAT. This pool summary reveals:

- The name of each available job pool (e.g., pbatch).
- The total available memory in each such pool (e.g., 1536Mb).
- Status information on each pool's nodes (CPUs/node; total, usable, and free node counts).
- Other relevant pool traits if any have been programmed.

SPJSTAT works with both LoadLeveler and SLURM as underlying batch systems on LC's AIX machines.

Establishing Authorization

For POE users, establishing authorization means insuring that you are permitted to run parallel jobs on the nodes you intend to use. On some SP systems this requires a fair amount of personal preparation, usually making and spreading appropriate .rhosts files to every node involved.

On LLNL's SP systems, however, AIX manages authorization and the system administrators have already done all necessary preparation. You can examine the file /etc/hosts.equiv (by running MORE, for example) on your SP login node and see a list of nodes (basically, all nodes) on which you have been preauthorized to run jobs.

Compiling and Linking Parallel Programs

Relevant Compilers

LOCATIONS.

POE provides parallel compiler scripts that automatically link in the necessary Parallel Environment libraries and then call the appropriate serial or threaded AIX compiler. This simplifies your use of the partition manager and the message-passing interface (MPI) library on LLNL's SP systems. Also, starting in 2002 IBM keeps its compilers in /usr/local/bin rather than in /usr/bin, so *avoiding* absolute pathnames when you invoke a compiler under AIX can avoid problems with unexpected compiler locations.

ALTERNATIVES.

The chart below shows the most important compiler scripts, organized by type of program compiled, available at LLNL. All of these compilers and scripts use the standard UNIX execute line sequence (name of program to run, then blank-delimited, hyphen-flagged options if any, then name of the source file to compile). Note that after January 26, 2000, the threaded MPI compilers became the default, so now all MPI compilations are automatically threaded on all LLNL SP systems open and secure (e.g., for eventual use with the gang scheduler).

Source to Compile -----	Compiler Invocation -----	Role Clarified -----
serial program	xlc (or cc)	ANSI standard C compiler
	xlC	C++ compiler
	gcc-2.95.3	GNU C compiler compatible with Totalview
	xlf (or f77)	Fortran 77 compatible code
	xlf90	Full Fortran 90 with IBM extensions
pthreads program	xlC_r	same as above
	xlC_r	but for parallel
	xlf_r	programs that use
	xlf90_r	pthreads
MPI with pthreads (*)	mpcc	parallel C programs with MPI (script)
	mpCC	parallel C++ programs with MPI (script)
	mpxlf	parallel Fortran 77 with MPI (script)
high- performance Fortran	xlhpf	high-performance Fortran 77
	xlhpf90	high-performance Fortran 90

(*)For MPI programs on LC's IBM machines, an alternative to using the native IBM compilers listed above (mpcc, etc.) is using the corresponding vendor-neutral MPICH compiler (called mpicc, etc.). A plain text file located on each machine at /usr/local/docs/MPI_Use_Summary gives the latest script names, version numbers, and execute lines for the available MPICH tools.

VERSIONS.

To allow you to move between compiler *versions* in an orderly way, LC puts the default version of each compiler under the name(s) shown in the chart above and puts the vendor's latest "new" version under newxlc, newxlc, and newxlf. With each advance in "new" versions, the former new version becomes the default in turn. On AIX machines at LC, multiple compiler versions also reside at

```
/usr/local/tools/compilers/ibm
```

in case application problems or new(er) compiler bugs cause you to revert temporarily to older versions of a compiler.

DYNAMIC LINKING.

Dynamically linked libraries are those not incorporated into your binary file at link time, but rather referenced by their location. All have .so suffixes (such as libnew.so). On LC AIX systems, the environment variable LIBPATH (unset by default) can contain an ordered, colon-delimited list of directory pathnames to search first for such dynamically linked libraries, before the standard library directories. LIBPATH thus fills the same role for AIX/POE that LD_LIBRARY_PATH fills for LC's Linux systems.

Relevant Compiler Options

All of the IBM SP compilers share (roughly) the same control options, and the most relevant options and their arguments are summarized in this section. See each compiler's MAN page for a complete list of its more obscure options and arguments.

Furthermore, all of these compilers have locally specified default options, set by the system administrator. The options you need most may have been already defaulted, so on each SP system you use you should (use MORE, for example, to) examine the configuration file from this list that is relevant to your compiler to discover the current local default options:

```
/etc/xlC.cfg  
/etc/xlf.cfg  
/etc/xlhpfcfg
```

(The poe.cfg configuration file used on some SP systems is disabled on those at LLNL.) Also, adequately managing floating-point exceptions and related NaN ("not a number") situations on IBM SP machines calls for quite a few strategic decisions about compiler options (like -qftrap) and system calls (like fp_trap). Consult the help file called /usr/local/docs/FPE_tips on each LC IBM system for a useful, detailed analysis of your alternatives.

These compiler options are among the most useful (those that begin with the -q prefix are called "keyword options," and they therefore may not fall where you expect in the alphabetical list):

-bmaxdata:*bytes*

specifies the data size if you expect to exceed the default data plus stack combination of 256 Mbyte.

-bmaxstack:*bytes*

specifies the stack size if you expect to exceed the default data plus stack combination of 256 Mbyte.

-c compiles only, without linking object files. The result is an output.o file.

-g generates the extra information needed by debuggers and some profiler tools.

-I *dirname* (uppercase eye) specifies a directory containing additional include files.

-L *dirname* (uppercase ell) specifies a directory (pathname) where additional libraries (named with the -l option, below) reside.

-l *key* (lowercase ell) specifies additional libraries to be searched, where you insert the string *key* and the compiler looks for a library called lib*key*.a.

-On (uppercase oh) selects a level of optimization, where *n* ranges from null (-O) for basic optimization through 2 to 3 (-O3) for "aggressive" optimization and 4 (-O4) for optimization tuned to the specific local platform.

- o (lowercase oh) specifies the name of the executable (a.out by default).
- p (also -pg) generates extra code to support profiling.
- qarch=*arch* specifies the architecture on which the executable will run, which can significantly improve performance at the expense of portability. Another option, called -qtune=*arch*, takes the same arguments as -qarch and serves just the same purpose. Possible values for *arch* include:
 - auto automatically detects the architecture of the compiling machine (use only if you compile and run on the *same* architecture).
 - com (the default) will run on any POWER or PowerPC hardware (such as the POWER5 machines UP, Tempest, and PU at LC).
 - pwr3 will run on POWER3 hardware machines (such as the former White and Ice at LC).
 - pwr4 will run on POWER4 hardware machines (such as UM and UV at LC).
- qlttrap checks for floating-point exceptions and generates a SIGTRAP signal if one occurs (but often with a high penalty on code performance). See the help file /usr/local/docs/FPE_tips on each LC IBM SP machine for suggestions and alternatives.
- qlanglvl=*level* specifies the language "level" (standard, standard subset, or standard superset) to check against for conformance, where the choices are ansi, saa, saa12, extended, and classic.
- qlist produces an object listing in the file output.lst.
- qsource produces a source listing in the file output.lst.
- qxref produces a cross-reference listing containing only referenced names.

Setting Up the Execution Environment

Under POE, your execution environment is managed by setting environment variables, as is typical on UNIX systems:

- Some are general UNIX variables (e.g., PATH).
- Some are specific to POE (e.g., MP_NODES), and those virtually all begin with the MP_ prefix.
- Some are effective only in some run situations (interactive), but are automatically ignored in others (batch). MP_RMPOOL is an example; others are marked with their description below.
- Some are enabled by LC's own Livermore Resource Management System (such as PSUB_JOBID) to provide a common environment for batch runs on *any* LCRM-managed LC machines (see the "Environment Variables" section of the LCRM (DPCS) Reference Manual (URL: <http://www.llnl.gov/LCdocs/dpcs>) for an explanatory list).
- Some facilitate job management and clean-up (e.g., LOADL_STEP_ID) but depend on IBM's LoadLeveler to function (see the "Between-Machine Script Differences" section (URL: <http://www.llnl.gov/LCdocs/ezjob/index.jsp?show=s5>) of EZJOBCONTROL for details). On AIX machines where LC has replaced LoadLeveler with SLURM as the underlying batch system, such specialized, dependent environment variables cease to function.

There are currently over 65 POE environment variables that influence the execution of your parallel jobs, and more are added in every new SP version.

The subsections of this section tell how to set these environment variables, make explicit the local LC default values, describe the role of the most important POE environment variables, and then summarize others that are relevant but devoted to more obscure or exotic uses. Users concerned about environment-variable differences (especially for switch or library control) between POE on IBM machines and the locally modified version of Linux (called CHAOS) that runs on LC's large, Intel-chip Linux clusters should check the environment-variable section of the manual CHAOS: Linux from Livermore. (URL: <http://www.llnl.gov/LCdocs/chaos>) Additional environment variables specifically for job management and resource allocation under Linux (CHAOS) are compared with their POE counterparts in the SLURM Reference Manual. (URL: <http://www.llnl.gov/LCdocs/slurm>)

Setting POE Environment Variables

You set POE environment variables using the standard commands in both the C (or TCSH) shell, namely

```
setenv varname varvalue
```

and in the the Bourne (or KSH) shell, namely

```
export varname=varvalue
```

For a general summary of how to use (set, unset, report) environment variables on LC's production machines, see the "Tools" [section](http://www.llnl.gov/LCdocs/ev/index.jsp?show=s2.3) (URL: <http://www.llnl.gov/LCdocs/ev/index.jsp?show=s2.3>) of LC's Environment Variables user manual.

You can set POE environment variables on any of four occasions:

- In response to any shell command prompt (e.g., just type: `setenv MP_PROCS 64`).
- By inserting commands within your shell's appropriate dot file (`.cshrc` or `.profile`).
- By running `SOURCE` on a script file before you execute your job (e.g., `source myvars`).
- By interspersing POE control flags with the arguments for your own program on its execute line, where such flags are made by combining a hyphen with the name of the environment variable you want to set (just the part following the `MP_` prefix). For example, you can set the `MP_PROCS` variable to 64 just for the duration of this job's run by typing

```
myprogram -procs 64 -myopt inputfile
```

For batch jobs, familiarity with the sequence in which LCRM automatically sets environment variables for each job that it runs may help you decide when (or if) you should intervene. See the "Batch-Job Environment Variables" [section](http://www.llnl.gov/LCdocs/ev/index.jsp?show=s3.4) (URL: <http://www.llnl.gov/LCdocs/ev/index.jsp?show=s3.4>) of LC's Environment Variables user manual for details. Note also that on AIX machines where SLURM rather than IBM's LoadLeveler is the underlying resource manager, SLURM will set its own family of environment variables that parallel many of those invoked by POE (or that would have been invoked by LoadLeveler). See the "Environment Variables" [section](http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.8) (URL: <http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.8>) of the SLURM Reference Manual for details.

Local Defaults

SET BY DEFAULT.

To provide a consistent, reliable environment for jobs, LC sets many environment variables to "appropriate" values by default on the production machines. On each machine, consult the file called `/etc/environment` to see the current default settings of all environment variables, including many not closely related to POE (such as the time zone). As a sample, the current POE environment variable values set by default on the AIX machine UP are:

Variable	Default value
MP_COREFILE_SIGTERM	no
MP_CPU_USE	unique
MP_EUILIB	us
MP_HOSTFILE	null
MP_INFOLEVEL	0
MP_LABELIO	yes
MP_PRIORITY	normalprio
MP_RESD	yes
MP_S_POE_AFFINITY	yes (enables TotalView)
MP_SHARED_MEMORY	yes

The role of (most of) these environment variables is explained (alphabetically by name) in the next subsection. For a more comprehensive (but hence more confusing) list of system-set environment variables on LC production machines (AIX, but also Linux), consult the "Kinds" [section](http://www.llnl.gov/LCdocs/ev/index.jsp?show=s3) (URL: <http://www.llnl.gov/LCdocs/ev/index.jsp?show=s3>) of the Environment Variables user manual.

IGNORED BY DEFAULT.

To understand your POE environment, you need to know about not only the variables set by default (above) but also about the environment variables that have no effect on batch (LoadLeveler or SLURM) jobs (those that are ignored if you use them in a LoadLeveler or SLURM job-control file). At LC, the following environment variables are never used by LoadLeveler or SLURM:

```
MP_ADAPTER_USE
MP_CPU_USE
MP_EUIDEVICE
MP_HOSTFILE
MP_NODES
MP_PMDSUFFIX
MP_PROCS
MP_RESD
MP_RETRY
MP_RETRYCOUNT
MP_RMPOOL
MP_SAVEHOSTFILE
MP_TASKS_PER_NODE
```

Basic POE Environment Variables

There are many POE environment variables, but by setting just a few basic ones (especially for interactive jobs) you can answer the three important questions:

- How many tasks/node (page 25) do I want?
- How will nodes be allocated (page 27) for my job?
- Which communications (page 28) protocol and network interface should I use?

Each subsection below describes the POE environment variables that answer one of these questions. A final subsection shows how the variables are typically used in combination to declare a (nondefault) operating environment for parallel jobs.

Tasks per Node

MP_PROCS (interactive jobs only) specifies the number of task processes for your parallel job (and hence, at one task per node, the size of your job's node partition). You may use **MP_PROCS** alone or with the other two variables in this section to specify how your tasks are loaded onto the SP's physical nodes. The default value is 1; the maximum varies from 128 to 2048 depending on the current version of POE (US protocol jobs can have up to 1024 tasks/job, while IP protocol jobs can have up to 2048 tasks/job. See the third subsection below on communications (page 28)).

MP_NODES (interactive jobs only) specifies the number of physical nodes on which to run your job's parallel tasks. You may use **MP_NODES** alone or with the other two variables in this section to specify how your tasks are loaded onto the SP's physical nodes.

MP_TASKS_PER_NODE

(interactive jobs only) specifies the number of tasks to run on each of the SP's physical nodes (up to a maximum of four). You may use **MP_TASKS_PER_NODE** alone or with the other two variables in this section to specify how your tasks are loaded onto the SP's physical nodes, but obviously the values that you assign must together make true the statement

$$\text{MP_TASKS_PER_NODE} * \text{MP_NODES} = \text{MP_PROCS}.$$

MP_TASK_AFFINITY

constrains each task of a parallel job, and all of its threads, to run within a specified "multichip module" (MCM, a node with 4 Power5 chips and 8 CPUs, plus local memory, an I/O interface, and a network adapter). Value choices are:

MCM allocates tasks round robin among the MCMs attached to the job by the resource manager. Tasks stay on the same MCM for the job's duration (recommended for nonOpenMP codes run on Purple).

SNI (User Space MPI jobs only) allocates tasks to the MCM in common with the first adapter assigned to the task by the resource manager.

<i>mcmlist</i>	specifies (by MCM numbers) a set of logical MCMs to which tasks are assigned round robin (not valid where LoadLeveler is also used).
-1	disables task affinity (use this or unset MP_TASK_AFFINITY for OpenMP jobs run interactively).

Node Allocation

MEMORY_AFFINITY

(AIX only) tells an AIX processor where to get memory, where the choices are:

- | | |
|-----------|--|
| MCM | makes both private and shared memory local to the processor's "multichip module" (MCM, see MP_TASK_AFFINITY (page 25) above for details). This choice is recommended for MPI jobs on Purple. |
| SHM=RR | stripes both System-V and POSIX real-time memory across multiple MCMs (only works for 64-bit systems). |
| LRU=EARLY | starts the LRU daemon using local memory as soon as its low threshold is reached, without waiting for all system pools to reach their low thresholds also. |

MP_RESD (interactive jobs only) specifies whether or not LoadLeveler (or SLURM on PU) or the individual user should allocate nodes to this job (ignored for batch jobs), where the choices are:

- | | |
|-----|---|
| YES | has LoadLeveler allocate the nodes (called nonspecific node allocation, this is the default at LC). |
| NO | has the user allocate the nodes (called specific node allocation, not used on LC machines). |

MP_RMPOOL (interactive jobs only) when LoadLeveler performs nonspecific node allocation, specifies the SP-system pool number from which the allocated nodes should be drawn. Use the [JS command](#) (page 12) to discover the pool numbers (and names) currently on your system. On AIX machines at LC where SLURM has *replaced* LoadLeveler as the underlying batch system, you must use the name of the desired node pool (e.g., pdebug) rather than its number to select it.

MP_HOSTFILE

(not used on LC SP systems) specifies the name of a file that contains the domain names of each node to use during specific node allocation (set to NULL by default at LC).

Task Communications

MP_BULK_MIN_MSG_SIZE

(default is 150000) specifies the message size in bytes above which bulk (AIX remote direct memory access, RDMA) transfers can begin (but see also MP_USE_BULK_XFER below). You can specify any value greater than or equal to 4000 (for Purple, 131200 is recommended for MPI optimization).

MP_EUILIB specifies which of two protocols should be used for task communications, where the choices are:

- | | |
|----|---|
| US | selects User Space protocol, the default on LC SP systems, and the faster of the two choices. |
| IP | selects Internet Protocol. |

MP_EUIDEVICE

(interactive jobs only) specifies which network adapter to use for communication among the SP nodes, where the choices are:

- | | |
|------|---|
| css0 | (css-zero) specifies the high-performance switch, the default value on LC SP systems and the best choice for most jobs. |
| en0 | (en-zero) specifies ethernet. |
| fi0 | (fi-zero) specifies FDDI. |
| tr0 | (tr-zero) specifies token ring. |

MP_POLLING_INTERVAL

specifies the polling interval in microseconds, that is, the delay until an MPI pthread wakes up to check for arriving messages. For US protocol the default is 400000 and for IP protocol the default is 180000 (see also MP_EUILIB above). On Purple, codes that are *not* interrupt driven (do *not* use MPI-IO, 1-sided MPI calls, parallel ESSL, or MP_CSS_INTERRUPT=YES) can optimize performance by setting MP_POLLING_INTERVAL to a value between 30000000 and 60000000 (30 to 60 seconds).

MP_USE_BULK_XFER

(default is NO) enables (YES) or disables (NO) reduced-overhead message transfer using AIX remote direct memory access (RDMA). If YES, recommended for optimization on Purple, then messages with length greater than MP_BULK_MIN_MSG_SIZE use bulk-transfer RDMA, while those shorter than that threshold still use packet-mode transfer. IBM warns that not all communication libraries honor this environment variable.

Examples Using Basic POE Variables

POE environment variables are usually set in related groups to control the execution of interactive parallel jobs. Here is a typical cluster of variables (set within the C shell) to perform nonspecific node allocation of 4 tasks to any combination of nodes from one pool:

```
setenv MP_PROCS 4
setenv MP_RMPOOL 0

setenv MP_RESD yes
setenv MP_HOSTFILE "null"
setenv MP_EUILIB us
setenv MP_EUIDEVICE css0
```

Note that the last four settings (selecting User Space protocol and the high-performance switch) are LC defaults that you therefore can omit.

Here is another example in which a cluster of POE environment variables performs nonspecific node allocation of 4 physical nodes, each executing 4 tasks (to run 16 total tasks):

```
setenv MP_NODES 4
setenv MP_TASKS_PER_NODE 4
setenv MP_RMPOOL 0

setenv MP_RESD yes
setenv MP_HOSTFILE "null"
setenv MP_EUILIB us
setenv MP_EUIDEVICE css0
```

Once again, the last four settings are defaults at LC.

Other POE Environment Variables

This list describes some potentially useful POE environment variables beyond the basic set covered in the previous section. They appear in alphabetical order, with usage restrictions (e.g., interactive only) and LC default values included whenever they apply. For an exhaustive (but hence confusing) annotated list of every POE environment variable, see the open web page at:

<http://www.llnl.gov/asci/platforms/bluepac/poe.envvars.html>

See the [Pthreads Overview \(for LC\)](http://www.llnl.gov/LCdocs/pthreads) (URL: <http://www.llnl.gov/LCdocs/pthreads>) for advice on using POSIX threads and the half dozen environment variables introduced here that tune pthreads performance under POE. On LC AIX machines where SLURM has replaced LoadLeveler as the underlying resource manager, SLURM's many environment variables will also affect how your job behaves. For most, see the relevant [section](http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.8) (URL: <http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.8>) of the SLURM Reference Manual, but the especially relevant SLURM_NETWORK variable is described in its alphabetical place in this section (below).

AIXTHREAD_COND_DEBUG

(default is OFF) toggles keeping a list of (Pthreads) condition variables to assist with debugging threaded programs.

AIXTHREAD_MINKTHREADS

overrides the AIXTHREAD_MNRATIO environment variable (next). This allows you to manually specify the minimum number of active kernel threads (default follows from MNRATIO). The library scheduler will not reclaim kernel threads below this number.

AIXTHREAD_MNRATIO

specifies the ratio of pthreads (M) to kernel threads (N). AIXTHREAD_MNRATIO is examined when the system creates a pthread to determine if a kernel thread should also be created to maintain the correct ratio. You can set this environment variable by supplying a value of the form

$$p:k$$

where k is the number of kernel threads the system uses to handle p (user) pthreads. You may specify any positive integer for p and k, but these values are used in a formula that employs integer arithmetic and this results in the loss of some precision when big numbers are specified. (See also AIXTHREAD_MINKTHREADS, above.)

Defaults:

If k is greater than p, the ratio is treated as 1:1.

If you specify no value, the default depends on the default contention scope.

If system scope contention is the default, the ratio is 1:1.

If process scope contention is the default, the ratio is 8:1.

AIXTHREAD_MUTEX_DEBUG

(default is OFF) toggles keeping a list of (Pthreads) active mutexes to assist with debugging threaded programs.

AIXTHREAD_RWLOCK_DEBUG

(default is OFF) toggles keeping a list of (Pthreads) read-write locks to assist with debugging threaded programs.

AIXTHREAD_SCOPE

sets the contention scope of pthreads created using the default pthread attribute object (for background on contention scope, see the "Alternative Thread Implementation Models" section of the Pthreads Overview (for LC) (URL: <http://www.llnl.gov/LCdocs/pthreads>). You can specify either of two exclusive values for this variable:

P indicates process scope (the default).

S indicates system scope.

AIXTHREAD_SLRATIO

determines the number of kernel threads used to support local pthreads sleeping in the library code while awaiting a pthread event, for example, attempting to obtain a mutex (discussed in the "Synchronization" section of the Pthreads Overview (for LC) (URL: <http://www.llnl.gov/LCdocs/pthreads>)). The reason to maintain kernel threads for sleeping pthreads is that, when the awaited pthread event occurs, the pthread will immediately need a kernel thread to run on. Using a kernel thread that is already available is more efficient than creating a new kernel thread after the event has taken place.

You can set this environment variable by supplying a value of the form

$k:p$

where k is the number of kernel threads to reserve for every p sleeping (user) pthreads. **WARNING:** the relative positions of k and p are reversed here from the ratio used to assign a value to AIXTHREAD_MNRATIO. You may specify any positive integer for p and k, but these values are used in a formula that employs integer arithmetic and this results in the loss of some precision when big numbers are specified. (See also AIXTHREAD_MINKTHREADS, above.)

Defaults:

If k is greater than p, the ratio is treated as 1:1.

If you specify no value, the default ratio is 1:12.

LDR_CNTRL toggles the use of large (16-Mbyte) rather than small (4-kbyte) memory pages by executables running on AIX machines (such as LC's UM, UV, or UP) where large memory pages account for most of the memory on each compute node. See also the "Enabling Large-Memory Pages" [section](#) (page 38) below for more background and the **MP_TLP_REQUIRED** variable later in this section for AIX responses to running without large pages enabled. The choices are:

LARGE_PAGE_DATA=N

(default) allows use of only small pages, which can cause excessive swapping and system crashes (and certainly code termination when paging space is exhausted).

LARGE_PAGE_DATA=Y

allows use of large (as well as small) memory pages. A [section](#) (page 38) below explains the benefits of this alternative.

LARGE_PAGE_DATA=M

("mandatory" mode) specifies that programs use *only* large memory pages. Once large pages are exhausted, the code terminates.

LIBPATH (AIX only; for Linux/CHAOS use **LD_LIBRARY_PATH**) specifies a colon-delimited list of directories for the linker to check (in order) to find dynamically linked libraries. At LC, **LIBPATH** is unset by default.

MP_ADAPTER_USE

(interactive jobs only) specifies whether or not your job is willing to share a node's switch adapter with other jobs, where the choices are **DEDICATED** (the default at LC and for all US communications) or **SHARED** (the nonLC default for IP communications).

MP_CHILD is a read-only variable set by POE for each task to contain its unique taskid (0 through **MP_PROCS**-1). Your batch script can query **MP_CHILD** to discover the identity of each task if needed.

MP_CMDFILE

specifies a POE commands file used to load programs onto the nodes of your partition instead of accepting commands from UNIX standard input (any pathname is a valid value). Generally, **MP_CMDFILE** is used only for multiple program multiple data jobs (i.e., when **MP_PGMODEL**=mpmd).

MP_COREFILE_FORMAT

(corresponds to the POE -corefile_format flag on AIX machines) specifies the format for corefiles generated when any of your processes terminate abnormally (this variable is especially suited for managing parallel programs). Possible values are:

[unset]	generates a standard AIX corefile for abnormal termination.
STDERR	sends the corefile contents to standard error for abnormal termination.
<i>any.other.string</i>	generates a lightweight corefile for abnormal termination. The string CORE.LIGHT, and hence this format alternative, is the default on LC's AIX machines.

MP_COREFILE_SIGTERM

(default is NO) prevents unexpected and undesirable core dumps that otherwise occur when you kill a parallel job (using PRM) or when a parallel job exits by calling MPI_Abort. Value YES allows such core dumps.

MP_CPU_USE

(interactive jobs only) specifies whether or not your job is willing to share a node's CPU with other jobs, where the choices are UNIQUE (no node sharing, the default at LC and in general for nonspecific node allocation of US communication jobs) or MULTIPLE (the nonLC default for IP jobs).

MP_EUIDEVELOP

controls the checking that MPI does during program execution, which may be helpful during program development. Valid values include YES, NO (the default), DEBUG, and NOC, where the last two toggle parameter checking and may significantly affect code performance.

MP_INFOLEVEL

specifies the level of messages reported as an aid to debugging, where the choices are

- 0 (error messages only),
- 1 (warning and error, the default at LC),
- 2 (informational, warning, and error),
- 3 (above plus diagnostic),
- 4, 5, 6 (above plus more elaborate diagnostic messages used by the IBM Support Center).

MP_LABELIO

specifies whether or not output from the parallel tasks is labeled by taskid, where the choices are YES (the default at LC) or NO.

MP_PGMODEL

specifies your programming model, where the choices are SPMD (single program multiple data, the default) or MPMD (multiple program multiple data, which enables you to load different executables individually on different nodes of your partition usually by using MP_CMDFILE to specify a suitable command file).

MP_RETRY (interactive jobs only) specifies the period (in seconds) between processor node allocation retries if there are not enough processor nodes immediately available.

MP_RETRYCOUNT

(interactive jobs only) specifies the number of times that the Partition Manager will attempt to allocate processor nodes before returning without running your program.

MP_SAVEHOSTFILE

(interactive jobs only) specifies the name of a file to be created by the Partition Manager and used to store the names of the hosts on which your POE job actually ran (a possible aid for debugging).

MP_SHARED_MEMORY

specifies whether tasks running on the same node should use shared memory (yes, the default) or the SP switch (no) for MPI message passing. Shared memory is faster for some codes, but the overhead may decrease the performance of others.

MP_SINGLE_THREAD

is an MPI library optimization flag. The choices are NO (the LC default), which assumes multiple message-passing threads and can improve library performance, or YES, which prevents your program from using MPI-IO.

MP_STDOUTMODE

manages the standard output from your parallel tasks, where the choices are UNORDERED (the default, and often required if you want to see the prompts from an interactive program as it runs, this has all tasks write output data to STDOUT asynchronously) or ORDERED (this has each task write its output to its own buffer, then later flushes all task buffers in task order to STDOUT).

MP_SYNC_ON_CONNECT

reveals nodes or tasks that cannot communicate (by trying an all-to-all synchronization as soon as the MPI library initializes, and looking for error messages about nonresponding tasks). But for large task counts setting this to yes (previous LC default) sometimes caused a timeout and job failure right at the start. LC now speeds job startup by setting MP_SYNC_ON_CONNECT=no by default.

MP_TIMEOUT

(no corresponding command-line flag) specifies how long (any time interval greater than 0.15 seconds) POE waits before abandoning an attempt to connect to remote nodes, and how long the communication subsystem waits for a connection to open during message-passing initiation (see MP_SYNC_ON_CONNECT above). With the "DCE and compatibility" SP security method, you may need to increase MP_TIMEOUT to allow time for the DCE servers to respond.

MP_TLP_REQUIRED

controls the system response if you run any MPI application *without* enabling large-memory pages. The reasons for using large pages on LC AIX machines are explained in the "Enabling Large-Memory Pages" [section](#) (page 38) below. The environment variable that toggles large-page use is LDR_CNTRL, explained above in this section. What happens if an MPI code tries to run without large pages depends on the value of MP_TLP_REQUIRED, where the three alternatives are:

WARN (default) causes AIX to issue a warning message when the MPI application starts and again once for every task launched. The message text is:

```
0:ATTENTION: 0031-522 current process
does not use large pages, continuing.
```

The application continues to run but this may cause it or the entire cluster to fail later, disrupting the work of all users. If you receive this warning message, you should stop your own job and rerun it with large pages enabled.

KILL terminates any MPI application that starts to execute without large pages (useful as a strong safeguard against flawed runs).

(unset) avoids both termination and warnings (intended only for those very unusual MPI situations that cannot use large pages for technical reasons and that have system-administrator permission to run with only small pages).

MP_TMPDIR specifies the location of temporary disk space for your job (the default at LC is /var/tmp, or /usr/tmp, which is just a link to /var/tmp).

MP_WAIT_MODE

specifies how a blocking MPI process will share computing resources, where the possible values are:

POLL (default) has the receiving thread actively poll for incoming messages. If tasks for several distinct jobs share a node, this might not be the best choice.

SLEEP has the receiving thread "sleep," and thus remove itself from the active dispatching queue if it has no work to do.

YIELD has the receiving thread stay in the queue but yield the processor if it has no work to do.

PCS_TMPDIR

(optional, if enabled by system administrator) not strictly speaking a POE variable, but available as a convenience to POE jobs, this contains the location (if any) of a specific temporary directory that is automatically created by LCRM for each batch job when the job begins and is automatically purged as soon as the batch job ends.

RT_GRQ

specifies whether the run queues for threads are assigned to a specific processor (off, the default) or managed globally (on). Some codes run faster if you switch this variable to ON (try comparative tests first).

SLURM_NETWORK

on AIX machines where SLURM has replaced LoadLeveler as the underlying resource manager, LCRM automatically sets this variable to specify four network features for each SLURM job step (which under AIX means for each POE invocation), using an argument with this sequential format:

network.[*protocol*],[*device*],[*adapteruse*],[*mode*]

where:

protocol specifies the network protocol (such as MPI).

device specifies the kind of switch used for communication (ethernet, FDDI, etc.), where the choices are the same abbreviation strings as the possible values of environment variable MP_EUIDEVICE (described above in the "Task Communications" [section](#) (page 28)).

adapteruse specifies whether (SHARED) or not (DEDICATED) your job is willing to share a node's switch adapter with other jobs (see also the corresponding POE environment variable MP_ADAPTER_USE, described above in this section).

mode specifies which of two protocols or modes should be used for task communications, where the choices are the same as the possible values of environment variable MP_EUILIB (described above in the "Task Communications" [section](#) (page 28)).

SPINLOOPTIME

(no default) specifies the number of times that the system will try to get a busy lock without taking a secondary action, such as calling the kernel to yield the processor. Manipulating SPINLOOPTIME can be helpful on SMP systems, where the lock might be held by another actively running pthread and will soon be released. On uniprocessor systems this value is ignored.

YIELDLOOPTIME

(no default) specifies the number of times that the system yields the processor when trying to acquire a busy mutex or spin lock (see the "Synchronization" section of the [Pthreads Overview \(for LC\)](#) (URL: <http://www.llnl.gov/LCdocs/pthreads>) for details)

before going to sleep on the lock. `YIELDLOOPTIME` can be helpful for complex applications where multiple locks are in use.

Enabling Large-Memory Pages on UM, UV, UP

On SCF, LC added the IBM/AIX machine called UM in late 2004 (with a duplicate called UV added in April, 2005). Both are 128-node POE machines that offer 8 Power4 CPUs per node (total 1024), where each node has 15 Gbyte of memory (16 Gbyte on UV). A similar 108-node POE (AIX) machine but with 8 Power5 CPUs per node (called UP) became generally available on OCF in August, 2005. On all three of these specific AIX computers, however, memory *configuration* is unusual, and often troublesome for "standard" jobs.

PROBLEM.

UM, UV, UP, and Purple too all use "large memory pages." Each compute node has 800 16-Mbyte memory pages, and these represent about 13 Gbyte of the 15 Gbyte total memory/node. Unfortunately, by default your code will make all of its memory requests to the remaining standard pool of 4-kbyte (small) memory pages. Since most of this 2-Gbyte standard pool is consumed by operating system requests and daemons, your code will quickly try using swap space after it exhausts the standard memory pool. LC terminates jobs that use excessive swap space to prevent system crashes. So default runs on UM, UV, UP, and Purple are likely to end badly, and they may even require administrator intervention to restart swamped nodes for other users. By default, MPI jobs run under AIX at LC without large-memory pages now receive this warning for every task attempted:

```
ATTENTION. 0031-522 current process
does not use large pages, continuing.
```

SOLUTION.

To compensate, you should enable your code to use large memory pages *before* you execute it on UM, UV, or UP. You can enable large-memory-page use in either of two ways:

- SET ENVIRONMENT VARIABLE.

Before execution, set the AIX LDR_CNTRL environment variable to LARGE_PAGE_DATA=Y with SETENV (for csh) or EXPORT (for sh or ksh) as shown here:

```
setenv LDR_CNTRL LARGE_PAGE_DATA=Y
```

or

```
export LDR_CNTRL="LARGE_PAGE_DATA=Y"
```

- RUN LDEDIT.

A special POE utility program called LDEDIT can modify the XCOFF header of any AIX executable file to declare that it is a "large-page" program. Running

```
ldedit -b lpdata
```

enables your executable to use large memory pages when they are available, yet still use standard pages otherwise.

Aside from this memory-management issue, running under POE on UM, UV, or UP is like running on any other LC IBM/AIX cluster. See also the entry for LDR_CNTRL in the "Other POE Environment Variables" [section](#) (page 30) above for a third alternative that *requires* a program to use only large memory

pages. See also `MP_TLP_REQUIRED` in that section for ways to control how AIX *responds* to attempts to run MPI applications that fail to enable large pages.

Running Under POE

Execute Lines

BACKGROUND.

For an explanation of the corresponding job-run alternatives on LC's Linux (CHAOS) clusters, see the SRUN subsections of the SLURM Reference Manual (URL: <http://www.llnl.gov/LCdocs/slurm>). Note that on some AIX machines (such as PU/Purple) LC actually uses SLURM rather than LoadLeveler for job management. On AIX machines that use SLURM instead of LoadLeveler you can submit your *script* to SLURM by executing SRUN, but within that script you should still use POE to execute your job *tasks*. SLURM under AIX does not directly support the Federation switch or proper use of IBM's MPI software. For a matrix showing the different job-control tools relevant to different resource-manager/operating-system combinations on LC clusters, see the "SLURM and Operating Systems" section (URL: <http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s2.3>) of the SLURM Reference Manual.

BATCH PARALLEL EXECUTION.

Most large parallel jobs run in batch mode, supervised by LC's Livermore Computing Resource Management (LCRM) system. This provides many automatic job scheduling and logging benefits, and makes the large PBATCH node pool (or other special pools) available to the job's parallel tasks (through PSUB's -pool option). See the EZJOBCONTROL Basic Guide (URL: <http://www.llnl.gov/LCdocs/ezjob>) and the LCRM (DPCS) Reference Manual (URL: <http://www.llnl.gov/LCdocs/dpcs>) for instructions on using LC's batch system effectively (and on checking LCRM job log files). IBM SP batch jobs are now scheduled by node pool rather than by job class (to support gang scheduling), a technical difference from other LC machines that most users should not notice. Remember also that on LC machines many POE environment variables are *ignored* by default during batch runs, so check the list of default settings (page 24) above before you spend time pointlessly trying to use them when they have no effect. (PSUB's former -noDFS option, once needed to avoid confusing warnings on some POE machines, became obsolete and irrelevant in January, 2003.) PSUB's -g ("geometry") option works on AIX machines only, regardless of whether they use SLURM or LoadLeveler.

INTERACTIVE PARALLEL EXECUTION.

Once you have written your program, compiled it, and set up the appropriate parallel environment using the POE variables, you can run it interactively simply by typing the name of the executable file. If you wish to set (or change) the values of some POE environment variables at run time, and just for the duration of the current run, you can use POE control flags on the execute line. Each POE control flag is made by combining a hyphen with the name of the environment variable you want to set (just the part following the MP_ prefix). For example, you can set the MP_PROCS variable to 64 just for the duration of this job's run by using the POE flag

```
-procs 64
```

These POE flags can be interspersed among the arguments of your own program, so that the general execute-line syntax becomes

```
executablename POE-control-flags your-arguments
```

and a typical example of using this feature is

```
myprogram -procs 64 -myopt inputfile
```

Remember that on LC AIX machines where SLURM has replaced LoadLeveler as the underlying batch system, you must use the node-pool name (e.g., -rmpool pdebug) rather than its number to select it with a POE flag.

DISTRIBUTED SERIAL EXECUTION.

To run a serial program (your own or a standard UNIX tool) across every node in your job's partition, use the special POE command that is literally called "poe," followed by the usual serial execute line (with its usual arguments, if any). Typical examples include:

```
poe myserialprog  
poe cp ~/input.file /var/tmp/input.file
```

PARALLEL FILE SYSTEMS.

If you make use of the parallel file systems unique to each LC POE (AIX) machine (called /p/gb1, etc.), you may need to coordinate your disk space needs with other users on the same machine. Every LC AIX machine therefore offers a system file called /usr/local/etc/pfs_status.*machinename* that reports for each available parallel file system its current total size, space already used, percentage used, percentage of possible inodes (roughly, files) used, and an ordered list of users and their current space usage (in both Tbytes and number of files).

Job Termination

For parallel BATCH jobs, LCRM handles job termination and error tracking. For parallel INTERACTIVE jobs that you run under POE, you can terminate the job (prematurely) in either of two ways.

First, you can terminate a running, interactive POE job by typing CTRL-C. This sends a SIGTERM signal to your home node's Partition Manager process, which in turn sends the termination signal to all remote tasks associated with that job. WARNING: type CTRL-C only once. Multiple CTRL-Cs may kill the Partition Manger itself, before it completes the shutdown of your remote processes. This can allow some of your processes to continue on remote nodes, tying up both those nodes and their switch adapters. These nodes will then remain unavailable for other jobs because LoadLeveler will think that you still need them.

Second, you can also invoke the POEKILL command to safely terminate all of your interactive POE tasks without typing CTRL-C. You execute POEKILL and use POE to distribute it among all nodes in your job's partition using the usual syntax, namely:

```
poe poekill yourprog -procs n -hostfile hlist
```

where *n* is the number of task processes in your parallel job and *hlist* is the name of a file containing a list of all the nodes where your tasks are running. POEKILL is seldom needed to terminate jobs at LC.

Pitfalls

New versions of AIX or other system changes sometimes bring with them unexpected or easily overlooked pitfalls that thwart effective use of POE. This section catalogs the known POE pitfalls as an aid to avoiding them, or at least debugging them afterwards.

AIX 5.1/4.3 INCOMPATIBILITIES.

LC installed AIX version 5.1 on all of its production IBM machines between the summer of 2002 and March, 2003. Because of changes in the `mpich.h` header file and in the MPI libraries between versions, MPI codes compiled or linked under AIX 5.1 will *not* work properly if run under AIX 4.3 (elsewhere). Likewise, 64-bit code compiled or linked under AIX 5.1 will not run under AIX 4.3 (and conversely, 64-bit code compiled or linked under AIX 4.3 will not run under AIX 5.1).

KCC LINK PROBLEMS.

The `newmpKCC` and `newmpKCC_r` compilers (both version 4.0f of KCC) on IBM AIX systems normally link to a shared library version of the KCC C++ library. This library conflicts with the IBM C++ shared library used by the MPI implementation under AIX 5.1. The typical result is illegal-instruction errors when your application uses `COUT`, `CERR`, or `CLOG`, or when MPI uses these routines in `MPI_Finalize`.

The workaround is to link to KCC's C++ library *statically*, by relinking with the option

```
--static_libKCC
```

added to your `newmpKCC` link execute line. Only relinking (not recompiling) is needed. The linker will emit a large number of "duplicate symbol" warnings for the C library, which tests have shown to be harmless for application codes under both AIX 5.1 and AIX 4.3. Note that this is a *run-time* problem, so codes executed on any AIX 5.1 system will need to be relinked with the static option above even if the link was originally done on an AIX-4.3 system.

OPEN/SECURE NETWORK TEST.

If you need your code to behave *differently* under AIX depending on its network environment (OCF or SCF), you can now query a line in the system file called `/etc/home.config` to discover the current environment. Every LC machine (including all AIX machines) offers a public `/etc/home.config` file, in which one line contains the keyword `NETWORK` (in uppercase) followed by one of two network values (`ocf` or `scf`, in lowercase). Your code can reveal the content of this line, for use in conditional tests, by executing

```
grep NETWORK /etc/home.config
```

AVOIDING BIG PROFILER OVERHEAD.

Once you start testing an MPI-based parallel version of your code, you can benefit from using `mpiP`, a "lightweight profiling library" specifically for MPI applications. The `mpiP` profiler on LC's AIX (IBM) machines:

- has been customized to handle long file names and function names,
- generates much less overhead, less between-task communication, and less total data than most heavy-duty tracing tools, and
- has a local documentation file available on each AIX node at `/usr/local/tools/mpiP/README`.

ENABLING LARGE-MEMORY PAGES.

On LC's SCF UM, UV, and UP machines, each node has 800 16-Mbyte "large-memory" pages, but only a 2-Gbyte pool of 4-kbyte ("small-memory" or standard) pages. Unfortunately, by default codes try to use only the small pages, which can cause excessive swapping and system crashes. You must explicitly enable your executable to use UM's available large-memory pages. You can either set environment variable LDR_CNTRL to the value LARGE_PAGE_DATA=Y or use the -b lpdata option of the special POE tool called LDEDIT. For details on what to do and why it matters, see the "Enabling Large-Memory Pages" section (page 38) in this manual.

TOGGLING REMOTE DIRECT MEMORY ACCESS (RDMA).

On LC's SCF UM, UV, and UP machines *only*, a scheme to accelerate MPI communication called "remote direct memory access" (RDMA) is enabled by default. But RDMA requires some setup time, so it may actually slow code execution unless you transfer a large amount of data. To compare how your code performs with and without RDMA you need a way to disable it. Starting in April, 2005, LCRM's PSUB utility (for submitting batch jobs) offers a -nobulkxfer option that works only on UM, UV, and UP and that disables RDMA. See also the RDMA-threshold discussion involving environment variables MP_BULK_MIN_MSG_SIZE and MP_USE_BULK_XFER in the "Task Communication" section (page 28) above.

DFS SUPPORT ENDS UNDER AIX 5.3.

As LC installed AIX version 5.3 on its production IBM machines in 2006, each one in turn lost its ability to mount the DFS (Distributed File Service) file system. If you have resources stored on DFS and need to move them elsewhere, you must log on to one of LC's smaller, older machines that will never be upgraded to AIX 5.3 (such as ICE or DEFROST) to be able to copy your DFS files. Also, consider using HTAR (URL: <http://www.llnl.gov/LCdocs/htar>).

Parallel Performance Benchmarks

SPHINX:

LLNL's Center for Applied Scientific Computing (CASC) now provides as publicly downloadable code a C-language integrated microbenchmark suite to conduct performance tests of every major variety of parallelized program on many different platforms (IBM, SGI, and Sun). LLNL's parallel benchmark suite, called Sphinx, has these features:

- Accesses each test action (such as message pingpong) through a function pointer, allowing different threads or tasks to execute different functions at once. This supports measurement of highly complex parallel actions.
- Times repeated calls (iterations) of each test action, stopping either when the standard deviation of the repetitions becomes less than a user-specified percentage of their mean or, if that never happens, after a user-specified maximum number of repetitions.
- (Optionally) corrects for test-suite ("harness") overhead and automatically warns if that overhead exceeds the measurement value of the test.
- Is highly portable (vendor-dependent binding of POSIX threads (pthreads) to processors is the chief threat to Sphinx portability).
- Covers (with suitable adaptations) pthreads (within-process parallelization), MPI (among-process parallelization), and OpenMP performance testing. Documentation at the Sphinx web site (below) specifically explains which tests apply to which features of the three approaches to parallelization.

Sphinx is available to the public at this open URL:

<http://www.llnl.gov/CASC/sphinx>

This Sphinx web site (UCRL-CODE-99026) provides all needed usage information and relevant files, including:

- A descriptive inventory of every file in the current Sphinx distribution, which includes every input file for the ASC milepost tests.
- Build and execution instructions for the test suite.
- Input file format and the input modes that Sphinx accepts.
- Output file format and the four Sphinx output streams.
- Specific test descriptions and allowed independent variables.
- References to (and in some cases even the full text of) published papers that present and discuss Sphinx results.

IBM PERFORMANCE PROBLEMS:

As of April, 2002, LC's massively parallel IBM computers sometimes showed significant performance problems for MPI programs, especially for those programs that (heavily) use library routines MPI_ALLREDUCE or MPI_BARRIER. One production physics code where MPI_ALLREDUCE was algorithmically expected to use about 1.9% of the total run time, for example, actually spent 90% of its MPI time and 30% of its physics time just executing MPI_ALLREDUCE. Extensive comparative testing by LC staff members suggests that the following kinds of codes are most susceptible to these serious performance problems:

- Parallel codes with fine-grained parallelization,
- Hybrid codes that combine MPI with OpenMP or with POSIX threads (Pthreads), and
- Codes that make heavy use of MPI_ALLREDUCE or MPI_BARRIER.

Users who see (or who wish to avoid) these problems are urged to profile their codes by running /usr/local/mpiP and to read the "IBM Confidential" analysis available at (OCF, special password required, request from the LC Hotline):

<http://www-r.llnl.gov/icc/viewgraphs/viewgraphs02/apr02/jones/index.htm>

for more details and for a few suggestions to work around these problems. By summer, 2003, IBM and LLNL staff had agreed that latent serial steps deep within AIX parallel operations were a primary cause of these performance problems. For an updated analysis and repair strategy, see

<http://www-r.llnl.gov/icc/viewgraphs/viewgraphs03/aug/jones/jones.pdf>

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government thereof, and shall not be used for advertising or product endorsement purposes.

(C) Copyright 2007 The Regents of the University of California. All rights reserved.

Keyword Index

To see an alphabetical list of keywords for this document, consult the next section (page 50).

Keyword	Description
<u>entire</u>	This entire document.
<u>title</u>	The name of this document.
<u>scope</u>	Topics covered in this document.
<u>availability</u>	Where these programs run.
<u>who</u>	Who to contact for assistance.
<u>introduction</u>	Role and goals of this document.
<u>poe-parts</u>	Software components of POE.
<u>poe-roles</u>	Tasks that POE performs.
<u>terminology</u>	Parallel terms defined, diagramed.
<u>poe-constraints</u>	POE constraints on app programs.
<u>configuration</u>	LC's machine configuration(s).
<u>reporting-tools</u>	Local SP config reporting tools.
<u>js</u>	JS summarizes node pools.
<u>ju</u>	JU summarizes current node usage.
<u>jj</u>	JJ summarizes current job features.
<u>llstatus</u>	LLSTATUS reports node properties.
<u>spjstat</u>	SPJSTAT reports job status, pool features.
<u>authorization</u>	How LC (pre)authorizes node use.
<u>compiling</u>	Compiling parallel programs at LC.
<u>compilers</u>	Relevant LC parallel compilers compared.
<u>compiler-options</u>	Relevant LC compiler options compared.
<u>environment</u>	LC's parallel execution environment.
<u>setting-variables</u>	Ways to set POE environment variables.
<u>default-variables</u>	LC's defaults for POE env. variables.
<u>basic-variables</u>	Most important POE variables explained.
<u>task-variables</u>	How tasks are loaded onto nodes.
<u>allocation-variables</u>	Which nodes are allocated to jobs.
<u>communication-variables</u>	How job tasks will communicate.
<u>example-variables</u>	Sample joint use of POE env. variables.
<u>advanced-variables</u>	Useful but exotic POE vars. explained.
<u>large-memory-pages</u>	Enabling large memory pages if needed.
<u>poe-usage</u>	Running interactive POE jobs.
<u>poe-execute-lines</u>	How to run parallel and serial jobs.
<u>job-termination</u>	How to stop parallel jobs.
<u>pitfalls</u>	Known POE/AIX problems noted.
<u>benchmarks</u>	Sphinx benchmark codes for MPI, pthreads.

index

a

date

revisions

The structural index of keywords.

The alphabetical index of keywords.

The latest changes to this document.

The complete revision history.

Alphabetical List of Keywords

Keyword -----	Description -----
<u>a</u>	The alphabetical index of keywords.
<u>advanced-variables</u>	Useful but exotic POE vars. explained.
<u>allocation-variables</u>	Which nodes are allocated to jobs.
<u>authorization</u>	How LC (pre)authorizes node use.
<u>availability</u>	Where these programs run.
<u>basic-variables</u>	Most important POE variables explained.
<u>benchmarks</u>	Sphinx benchmark codes for MPI, pthreads.
<u>communication-variables</u>	How job tasks will communicate.
<u>compiler-options</u>	Relevant LC compiler options compared.
<u>compilers</u>	Relevant LC parallel compilers compared.
<u>compiling</u>	Compiling parallel programs at LC.
<u>configuration</u>	LC's machine configuration(s).
<u>date</u>	The latest changes to this document.
<u>default-variables</u>	LC's defaults for POE env. variables.
<u>entire</u>	This entire document.
<u>environment</u>	LC's parallel execution environment.
<u>example-variables</u>	Sample joint use of POE env. variables.
<u>index</u>	The structural index of keywords.
<u>introduction</u>	Role and goals of this document.
<u>jj</u>	JJ summarizes current job features.
<u>job-termination</u>	How to stop parallel jobs.
<u>js</u>	JS summarizes node pools.
<u>ju</u>	JU summarizes current node usage.
<u>large-memory-pages</u>	Enabling large memory pages if needed.
<u>llstatus</u>	LLSTATUS reports node properties.
<u>pitfalls</u>	Known POE/AIX problems noted.
<u>poe-constraints</u>	POE constraints on app programs.
<u>poe-execute-lines</u>	How to run parallel and serial jobs.
<u>poe-parts</u>	Software components of POE.
<u>poe-roles</u>	Tasks that POE performs.
<u>poe-usage</u>	Running interactive POE jobs.
<u>reporting-tools</u>	Local SP config reporting tools.
<u>revisions</u>	The complete revision history.
<u>scope</u>	Topics covered in this document.
<u>setting-variables</u>	Ways to set POE environment variables.
<u>spjstat</u>	SPJSTAT reports job status, pool features.
<u>task-variables</u>	How tasks are loaded onto nodes.
<u>terminology</u>	Parallel terms defined, diagramed.
<u>title</u>	The name of this document.
<u>who</u>	Who to contact for assistance.

Date and Revisions

Revision Date -----	Keyword Affected -----	Description of Change -----
13Aug07	<u>advanced-variables</u>	SLURM_NETWORK and cross refs added. MP_TLP_REQUIRED details revised. MP_EUIDEVELOP name fixed.
	<u>large-memory-pages</u>	Details updated, expanded.
04Apr07	<u>advanced-variables</u>	MP_TLP_REQUIRED added, referenced.
	<u>large-memory-pages</u>	Cross ref to MP_TLP_REQUIRED added.
17Jan07	<u>default-variables</u>	MP_S_POE_AFFINITY added.
	<u>task-variables</u>	MP_TASK_AFFINITY added.
	<u>allocation-variables</u>	MEMORY_AFFINITY added.
	<u>communication-variables</u>	Polling and RDMA controls added.
	<u>advanced-variables</u>	More LDR_CNTRL suboptions.
04Oct06	<u>setting-variables</u>	Cross ref added to SLURM env vars.
	<u>poe-execute-lines</u>	Cross ref added to tools/systems matrix.
19Jun06	<u>pitfalls</u>	DFS support ends with AIX 5.3.
	<u>terminology</u>	SLURM replaces LoadLeveler under AIX.
	<u>reporting-tools</u>	SINFO, SQUEUE in play under AIX.
	<u>environment</u>	LOADL_STEP_ID not with SLURM.
	<u>allocation-variables</u>	Pool names replace numbers with SLURM.
16Feb06	<u>compilers</u>	LIBPATH role explained.
	<u>setting-variables</u>	Cross ref to Env Var manual added.
	<u>default-variables</u>	Cross ref to Env Var manual added.
	<u>advanced-variables</u>	MP_COREFILE_FORMAT, 3 thread debug vars added.
	<u>poe-execute-lines</u>	SRUN, POE roles with SLURM clarified.
11Oct05	<u>compilers</u>	Version information added.
	<u>compiler-options</u>	Architecture suboptions updated.
	<u>terminology</u>	SLURM under AIX noted.
	<u>poe-usage</u>	SLURM under AIX noted.
	<u>default-variables</u>	Variables also ignored by SLURM.

07Sep05	<u>large-memory-pages</u>	Expanded analysis, covers UP too.
15Aug05	<u>poe-execute-lines</u>	Some batch features updated.
	<u>introduction</u>	POE machine list updated.
21Apr05	<u>spjstat</u>	TC2K retired, details removed.
	<u>large-memory-pages</u>	Same info now also applies to UV.
	<u>pitfalls</u>	Toggling RDMA on UM/UV added.
18Jan05	<u>large-memory-pages</u>	New section on large memory pages.
	<u>index</u>	New keyword for new section.
	<u>advanced-variables</u>	LDR_CNTRL added, cross refd.
	<u>pitfalls</u>	Large-memory-page issue noted.
04Oct04	<u>introduction</u>	Blue references deleted.
	<u>reporting-tools</u>	Blue references replaced in examples.
21Jul04	<u>spjstat</u>	Three roles contrasted, clarified.
	<u>index</u>	SPJSTAT roles updated.
02Mar04	<u>default-variables</u>	MP_COREFILE_SIGTERM added.
	<u>advanced-variables</u>	MP_COREFILE_SIGTERM explained.
27Oct03	<u>introduction</u>	Cross ref to SLURM manual added.
	<u>environment</u>	Comparison with SLURM vars. noted.
	<u>poe-usage</u>	Comparison with SRUN usage noted.
03Sep03	<u>benchmarks</u>	More on MPI performance problem.
	<u>pitfalls</u>	Lightweight profiler mpiP noted.
24Jun03	<u>poe-constraints</u>	MASS supplements LIBM.
	<u>poe-execute-lines</u>	Usage info for parallel file systems.
	<u>pitfalls</u>	OCF/SCF network test explained.
12Mar03	<u>introduction</u>	Cross ref to CHAOS manual added.
	<u>environment</u>	Cross ref to CHAOS manual added.
	<u>pitfalls</u>	New section on known problems.
	<u>index</u>	New keyword for new section.
21Jan03	<u>environment</u>	Special PSUB batch variables noted.
	<u>configuration</u>	Limits reporting clarified.
	<u>poe-usage</u>	Former -noDFS option now obsolete.
19Aug02	<u>compilers</u>	Warning to avoid absolute pathnames.
	<u>compiler-options</u>	QFLTTRAP and FPE_tips added.
	<u>default-variables</u>	POE default settings updated.
	<u>advanced-variables</u>	

		MP_SINGLE_THREAD added.
08May02	<u>benchmarks</u>	MPI/pthreads performance problem.
23Apr02	<u>compilers</u> <u>configuration</u>	GNU GCC added. Resource partition limits explained.
16Jan02	<u>introduction</u> <u>poe-constraints</u>	Finnish SP manual cited. ULIMIT report added.
26Nov01	<u>introduction</u>	AIX/Linux differences noted.
16Oct01	<u>compilers</u> <u>task-variables</u> <u>advanced-variables</u> <u>poe-usage</u>	Local MPICH instructions referenced. Details of use clarified. Details clarified, 1 more added. Warning added on ignored e.v.'s.
23Jul01	<u>advanced-variables</u>	Several LC default values changed. RT_GRQ added.
12Jun01	<u>poe-usage</u> <u>benchmarks</u> <u>index</u>	PSUB -noDFS now OCF default. New section on parallel benchmarks. New keyword for new section.
22Feb01	<u>advanced-variables</u>	Six pthreads env vars added.
09Jan01	<u>spjstat</u> <u>advanced-variables</u> <u>configuration</u> <u>poe-usage</u> <u>index</u>	New section on job-reporting tool. PCS_TMPDIR variable added. job.limits file noted. Node pool role noted. New keyword for new section.
26Oct00	<u>poe-usage</u>	Role of -noDFS PSUB option noted.
05Sep00	<u>introduction</u> <u>advanced-variables</u>	Cross ref to IBM Doc Dir added. Exhaustive var list cross referenced.
19Apr00	<u>availability</u> <u>default-variables</u> <u>advanced-variables</u>	New print instructions. MP_SYNC_ON_CONNECT added. MP_SYNC, MP_SHARED_MEMORY added.
15Feb00	entire	First edition of POE User Guide.

TRG (13Aug07)

UCRL-WEB-201527

LLNL Privacy and Legal Notice (URL: <http://www.llnl.gov/disclaimer.html>)

TRG (13Aug07) Contact on the OCF: lc-hotline@llnl.gov, on the SCF: lc-hotline@pop.llnl.gov